

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2023–2024 FALL SEMESTER**

**PROJECT-3 REPORT**

**ARAMA AĞAÇLARI, YIĞINLAR ve HASH TABLOSU: UNESCO M. ALANI BİLGİ SİSTEMİ**

**DELIVERY DATE**

22/12/23

**PREPARED BY**

05220000981, BURAK TAŞKIN

05210000278, İSMAİL ERKAN

İçindekiler

[1) UM AĞACI 4](#_Toc153463973)

[1.a UM\_Ağacı Ağacının Oluşturulması 4](#_Toc153463974)

[1.a.1 Kodlar 4](#_Toc153463975)

[1.a.2 Açıklama 5](#_Toc153463976)

[1.b Ağacın Derinliğini, Düğüm Sayısını ve Ağaçtaki Bilgileri Ekrana Listeleme 6](#_Toc153463977)

[1.b.1 Kodlar 6](#_Toc153463978)

[1.b.2 Ekran görüntüleri 7](#_Toc153463979)

[1.b.3 Açıklama 8](#_Toc153463980)

[1.c UM\_Alanlarının İsimlerinin Listelenmesi 9](#_Toc153463981)

[1.c.1 Kodlar 9](#_Toc153463982)

[1.c.2 Ekran görüntüleri 10](#_Toc153463983)

[1.d Özyineli olarak Dengeli Bir Ağaç Oluşturulması 10](#_Toc153463984)

[1.d.1 Kodlar 10](#_Toc153463985)

[1.d.2 Ekran görüntüleri 12](#_Toc153463986)

[1.d.3 Açıklama 13](#_Toc153463987)

[2) HASH TABLE 13](#_Toc153463988)

[2.a UM\_Alanı Nesnelerini Hash Table’a Yerleştirme 13](#_Toc153463989)

[2.a.1 Kaynak Kod 13](#_Toc153463990)

[2.a.2 Ekran Görüntüleri 14](#_Toc153463991)

[2.b Hash Table Güncelleme 14](#_Toc153463992)

[2.b.1 Kaynak Kod 14](#_Toc153463993)

[2.b.2 Ekran Görüntüsü 15](#_Toc153463994)

[2.b.3 Açıklama 15](#_Toc153463995)

[3) HEAP VERİ YAPISI (SINIFI) 16](#_Toc153463996)

[3.a Heap 16](#_Toc153463997)

[3.a.1 Kaynak Kod 16](#_Toc153463998)

[3.b Min Heap’e Yerleştirme 17](#_Toc153463999)

[3.b.1 Kaynak Kod 17](#_Toc153464000)

[3.c Min Heap Listeleme 18](#_Toc153464001)

[3.c.1 Kaynak Kod 18](#_Toc153464002)

[3.c.2 Ekran Görüntüsü 18](#_Toc153464003)

[3.c.3 Açıklama 18](#_Toc153464004)

[4) SIRALAMA 19](#_Toc153464005)

[4.a Sıralama Algoritmaları 19](#_Toc153464006)

[4.a.1 Kaynak Kod 19](#_Toc153464007)

[4.a.2 Açıklama 21](#_Toc153464008)

[4.b Zaman Karmaşıklığının Hesaplanması 21](#_Toc153464009)

[4.b.1 Açıklama 21](#_Toc153464010)

[5) Görselleştirme 22](#_Toc153464011)

[5.a Görselleştirmenin Etkisi 22](#_Toc153464012)

[5.a.1 Açıklama 22](#_Toc153464013)

[5.b Karşılaştırma Yapılması 22](#_Toc153464014)

[5.b.1 Açıklama 22](#_Toc153464015)

[6) Özdeğerlendirme Tablosu 23](#_Toc153464016)

# 1) UM AĞACI

//The platform, version, and programming language used

Inteljj, "19.0.1", JAVA

## 1.a UM\_Ağacı Ağacının Oluşturulması

### 1.a.1 Kodlar

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***package*** proje33333deneme;    ***import*** java.util.ArrayList; ***import*** java.util.List; ***import*** java.util.StringTokenizer;     ***public class*** UmAlani {     UmAlani left;  UmAlani right;  String name;  String city;  ***int*** year;  List<String> wordList ;  String text;     ***public*** UmAlani(String name, String city, ***int*** year, String text ){  ***this***.name = name;  ***this***.city = city;  ***this***.year = year;  ***this***.text = text;  wordList = ***this***.extractWords();  }   ***private*** List<String> extractWords(){   List<String> wordList = ***new*** ArrayList<>();  StringTokenizer tokenizer = ***new*** StringTokenizer(text, " ,.:");  ***while*** (tokenizer.hasMoreTokens()) {  String word = tokenizer.nextToken();  wordList.add(word);  }   ***return*** wordList;  }    ***@Override  public*** String toString() {  ***return*** "UmAlanı: " +  "name='" + name + '\'' +  ", City='" + city + '\''+  ", Year='" + year + '\''+  ", text='" + text + '\''  ;  }             } |

### 1.a.2 Açıklama

//Kullanılan veri yapılarını ve algoritmanın kısaca anlatımını burada gerçekleştiriniz

Alanlar (Fields):

left ve right: İkili arama ağacındaki sol ve sağ alt düğümleri gösteren referanslardır.

name: İş alanının adını temsil eden bir String.

city: İş alanının bulunduğu şehri temsil eden bir String.

year: İş alanının kuruluş yılını temsil eden bir tamsayı.

wordList: Bir metindeki kelimeleri içeren bir liste.

text: İş alanına ait metni temsil eden bir String.

Yapıcı Metot (Constructor):

Bir UmAlani nesnesi oluşturulurken, ad, şehir, yıl ve metin bilgileri ile başlatılır.

extractWords metodu kullanılarak metindeki kelimeler çıkarılır ve wordList listesine eklenir.

extractWords Metodu:

Metindeki kelimeleri çıkarmak için kullanılır.

StringTokenizer sınıfı, belirtilen ayırıcı karakterlere göre metni parçalarına ayırır.

Her bir kelime, wordList listesine eklenir ve bu liste geri döndürülür.

toString Metodu:

Nesnenin String temsilini döndüren toString metodu override edilmiştir. Bu metodun çıktısı, iş alanının adı, şehri, yılı ve metni içerir.

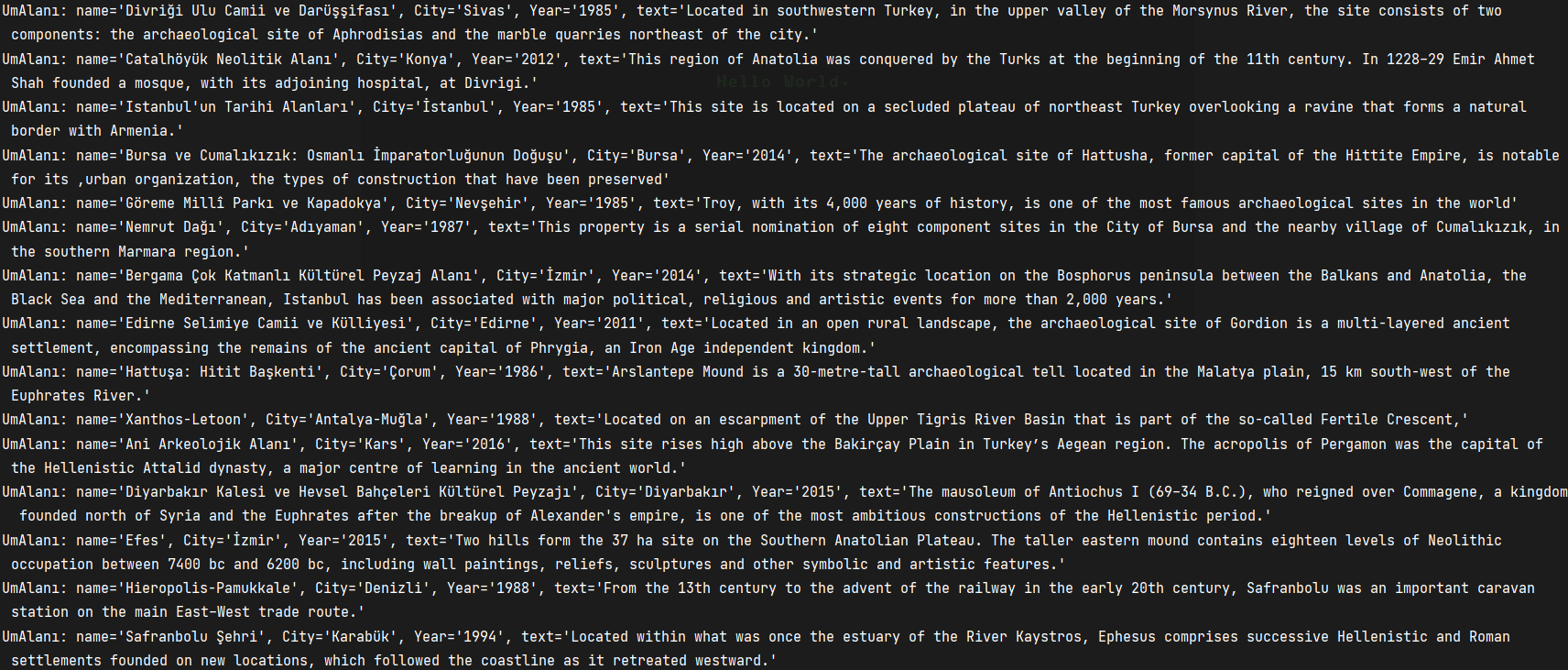
## 1.b Ağacın Derinliğini, Düğüm Sayısını ve Ağaçtaki Bilgileri Ekrana Listeleme

### 1.b.1 Kodlar

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***class*** TreeNode {  UmAlani data;  TreeNode left, right;   ***public*** TreeNode(UmAlani data) {  ***this***.data = data;  ***this***.left = ***this***.right = ***null***;  } }  ***public class*** BinarySearchAlphabetical {  TreeNode root;   ***public*** BinarySearchAlphabetical() {  root = ***null***;  }   ***void*** insert(UmAlani data) {  root = insertRec(root, data);  }   TreeNode insertRec(TreeNode root, UmAlani data) {  ***if*** (root == ***null***) {  ***return new*** TreeNode(data);  }   ***if*** (data.name.compareTo(root.data.name) < 0) {  root.left = insertRec(root.left, data);  } ***else if*** (data.name.compareTo(root.data.name) > 0) {  root.right = insertRec(root.right, data);  }   ***return*** root;  }   ***void*** levelOrderTraversal() {  ***int*** height = height(root);  ***for*** (***int*** i = 1; i <= height; i++) {  printGivenLevel(root, i);  }  }   ***int*** height(TreeNode root) {  ***if*** (root == ***null***) {  ***return*** 0;  } ***else*** {  ***int*** leftHeight = height(root.left);  ***int*** rightHeight = height(root.right);   ***return*** Math.max(leftHeight, rightHeight) + 1;  }  }   ***void*** printGivenLevel(TreeNode root, ***int*** level) {  ***if*** (root == ***null***) {  ***return***;  }  ***if*** (level == 1) {  System.out.println(root.data.toString());  } ***else if*** (level > 1) {  printGivenLevel(root.left, level - 1);  printGivenLevel(root.right, level - 1);  }  } } |

### 1.b.2 Ekran görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz



### 

### 

### 1.b.3 Açıklama

//Kullanılan veri yapıları ve algoritmanın kısaca anlatımını burada gerçekleştiriniz

TreeNode Sınıfı:

Bu sınıf, ikili arama ağacındaki düğümleri temsil eder.

Her düğüm, bir UmAlani nesnesini (data) içerir ve sol ve sağ alt ağaçları gösteren referanslara (left ve right) sahiptir.

Yapıcı metot, bir UmAlani nesnesi alır ve düğümü oluşturur.

BinarySearchAlphabetical Sınıfı:

Bu sınıf, ikili arama ağacını yönetir.

root değişkeni, ağacın kök düğümünü temsil eder.

insert metodu, ağaca bir düğüm eklemek için kullanılır. İkili arama ağacı özelliğini korumak için rekürsif bir yardımcı metot olan insertRec kullanılır.

levelOrderTraversal metodu, ağacı seviye seviye dolaşarak düğümleri ekrana yazdırır.

height metodu, bir düğümün yüksekliğini hesaplamak için kullanılır. Ağacın yüksekliğini bulmak için rekürsif bir yaklaşım kullanır.

printGivenLevel metodu, belirli bir seviyede bulunan düğümleri yazdırmak için kullanılır.

Algoritma Açıklamaları:

Düğüm Ekleme (Insertion): Yeni bir düğüm eklenirken, ağaç üzerinde gezilir ve uygun konum bulunur. Ekleme işlemi, düğümlerin UmAlani nesnelerinin name özelliğine göre alfabetik sıraya göre yapılır.

Seviye Sırasında Dolaşma (Level-Order Traversal): Ağaç seviye seviye dolaşılır. Her seviyede soldan sağa doğru ilerlenir ve düğümler ekrana yazdırılır.

Ağacın Yüksekliği (Tree Height): Her düğümün yüksekliği hesaplanır ve en sonunda ağacın yüksekliği bulunur. Yükseklik, sol ve sağ alt ağaçların yüksekliklerinin maksimumu alınarak hesaplanır.

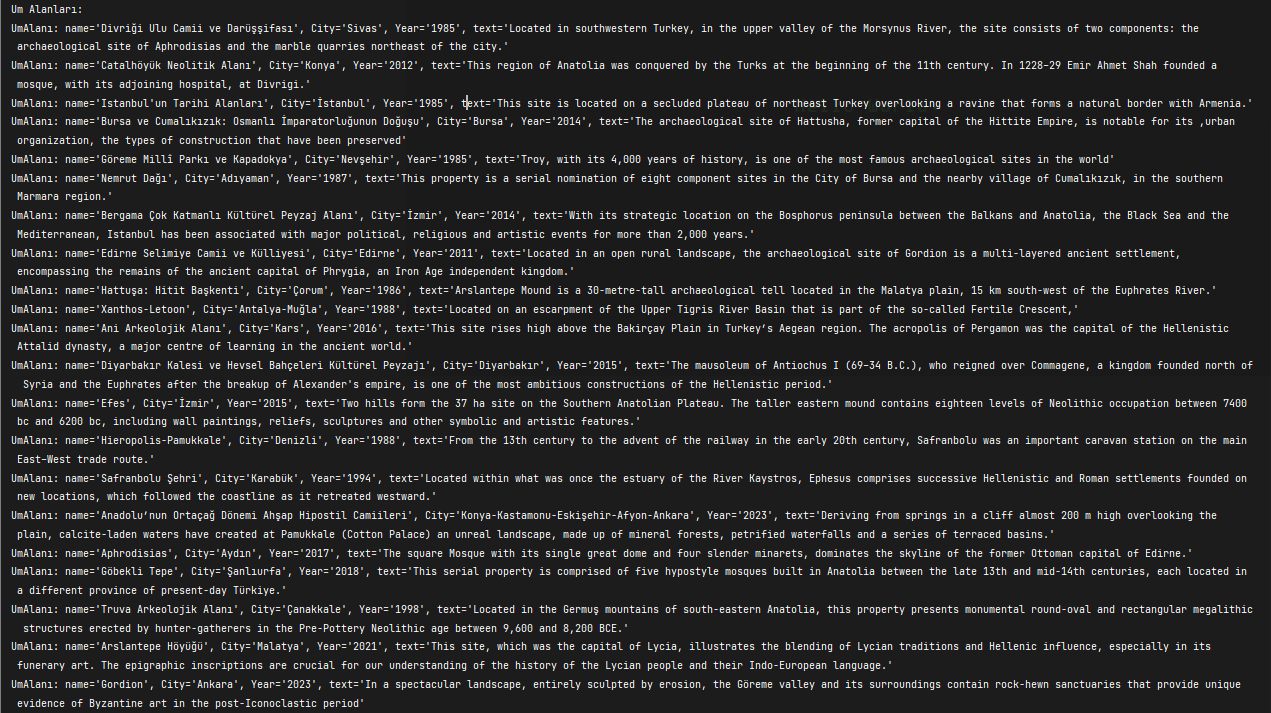
## 1.c UM\_Alanlarının İsimlerinin Listelenmesi

### 1.c.1 Kodlar

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***void*** printGivenLevel(TreeNode root, ***int*** level) {  ***if*** (root == ***null***) {  ***return***;  }  ***if*** (level == 1) {  System.out.println(root.data.toString());  } ***else if*** (level > 1) {  printGivenLevel(root.left, level - 1);  printGivenLevel(root.right, level - 1);  } }  ***void*** levelOrderTraversal() {  ***int*** height = height(root);  ***for*** (***int*** i = 1; i <= height; i++) {  printGivenLevel(root, i);  } } |

### 1.c.2 Ekran görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

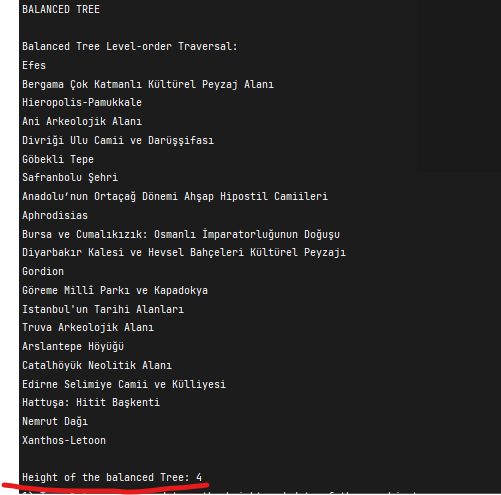


## 1.d Özyineli olarak Dengeli Bir Ağaç Oluşturulması

### 1.d.1 Kodlar

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***import*** java.util.\*;   ***public class*** BalancedBinaryTree {  TreeNode root;   ***public*** BalancedBinaryTree() {  root = ***null***;  }   ***void*** insert(UmAlani data) {  root = insertRec(root, data);  }   TreeNode insertRec(TreeNode root, UmAlani data) {  ***if*** (root == ***null***) {  ***return new*** TreeNode(data);  }   ***if*** (data.name.compareTo(root.data.name) < 0) {  root.left = insertRec(root.left, data);  } ***else if*** (data.name.compareTo(root.data.name) > 0) {  root.right = insertRec(root.right, data);  }   ***return*** root;  }   TreeNode sortedArrayToBST(UmAlani[] umAlanlari, ***int*** start, ***int*** end) {  ***if*** (start > end) {  ***return null***;  }   ***int*** mid = (start + end) / 2;  TreeNode newNode = ***new*** TreeNode(umAlanlari[mid]);   newNode.left = sortedArrayToBST(umAlanlari, start, mid - 1);  newNode.right = sortedArrayToBST(umAlanlari, mid + 1, end);   ***return*** newNode;  }   ***public void*** createBalancedTree(UmAlani[] umAlanlari) {  Arrays.sort(umAlanlari, Comparator.comparing(um -> um.name));  root = sortedArrayToBST(umAlanlari, 0, umAlanlari.length - 1);  }   ***void*** inOrderTraversal(TreeNode root) {  ***if*** (root != ***null***) {  inOrderTraversal(root.left);  System.out.println(root.data.name);  inOrderTraversal(root.right);  }  }   ***void*** levelOrderTraversal() {  ***if*** (root == ***null***) {  ***return***;  }   Queue<TreeNode> queue = ***new*** LinkedList<>();  queue.add(root);   ***while*** (!queue.isEmpty()) {  TreeNode tempNode = queue.poll();  System.out.println(tempNode.data.name);   ***if*** (tempNode.left != ***null***) {  queue.add(tempNode.left);  }   ***if*** (tempNode.right != ***null***) {  queue.add(tempNode.right);  }  }  }   ***int*** height(TreeNode root) {  ***if*** (root == ***null***) {  ***return*** 0;  } ***else*** {  ***int*** leftHeight = height(root.left);  ***int*** rightHeight = height(root.right);   ***return*** Math.max(leftHeight, rightHeight) + 1;  }  }   ***public int*** getHeight() {  ***return*** height(root);  } } |

### 1.d.2 Ekran görüntüleri

//

### 1.d.3 Açıklama

// insert metodu, ikili arama ağacına yeni bir düğüm ekler ve ağacın dengesini bozmaz.

createBalancedTree metodu, gelen diziyi sıralar ve sıralı diziyi kullanarak dengeli bir ikili arama ağacı oluşturur.

Ağacın sıralı veya seviye seviye gezilmesi, düğümlerin doğru sırayla ziyaret edilmesini sağlar.

height metodu, her düğümün yüksekliğini hesaplar ve en sonunda ağacın yüksekliğini döndürür.

# 2) HASH TABLE

// Inteljj, "19.0.1", JAVA

## 2.a UM\_Alanı Nesnelerini Hash Table’a Yerleştirme

### 2.a.1 Kaynak Kod

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***import*** java.util.ArrayList; ***import*** java.util.LinkedList; ***import*** java.util.Scanner;  ***public class*** DirectChaining {  LinkedList<String>[] hashTable;  ***int*** maxChainSize = 5;   DirectChaining(***int*** size) {  hashTable = ***new*** LinkedList[size];  }   ***public int*** modASCIIHashFunction(String word, ***int*** M) {  ***char*** ch[];  ch = word.toCharArray();  ***int*** i, sum;  ***for*** (sum=0,i=0; i<word.length(); i++) {  sum = sum + ch[i];  }  ***return*** sum % M;  }   ***public void*** insertHashTable(String word) {  ***int*** newIndex = modASCIIHashFunction(word, hashTable.length);  ***if***(hashTable[newIndex] == ***null***) {  hashTable[newIndex] = ***new*** LinkedList<String>();  hashTable[newIndex].add(word);  } ***else*** {  hashTable[newIndex].add(word);  }  }   ***public void*** displayHashTable() {  ***if*** (hashTable == ***null***) {  System.out.println("\nHashTable does not exists");  ***return***;  } ***else*** {  System.out.println("\n----------HashTable----------");  ***for*** (***int*** i=0; i<hashTable.length; i++) {  System.out.println("Index " + i + ", key:" + hashTable[i]);  }  }  } |

### 2.a.2 Ekran Görüntüleri

// Nesnelerin Hash Table’a Yerleştirildiğini Gösteren Konsol Çıktısı Ekran Görüntüsü metin, ekran görüntüsü içeren bir resim

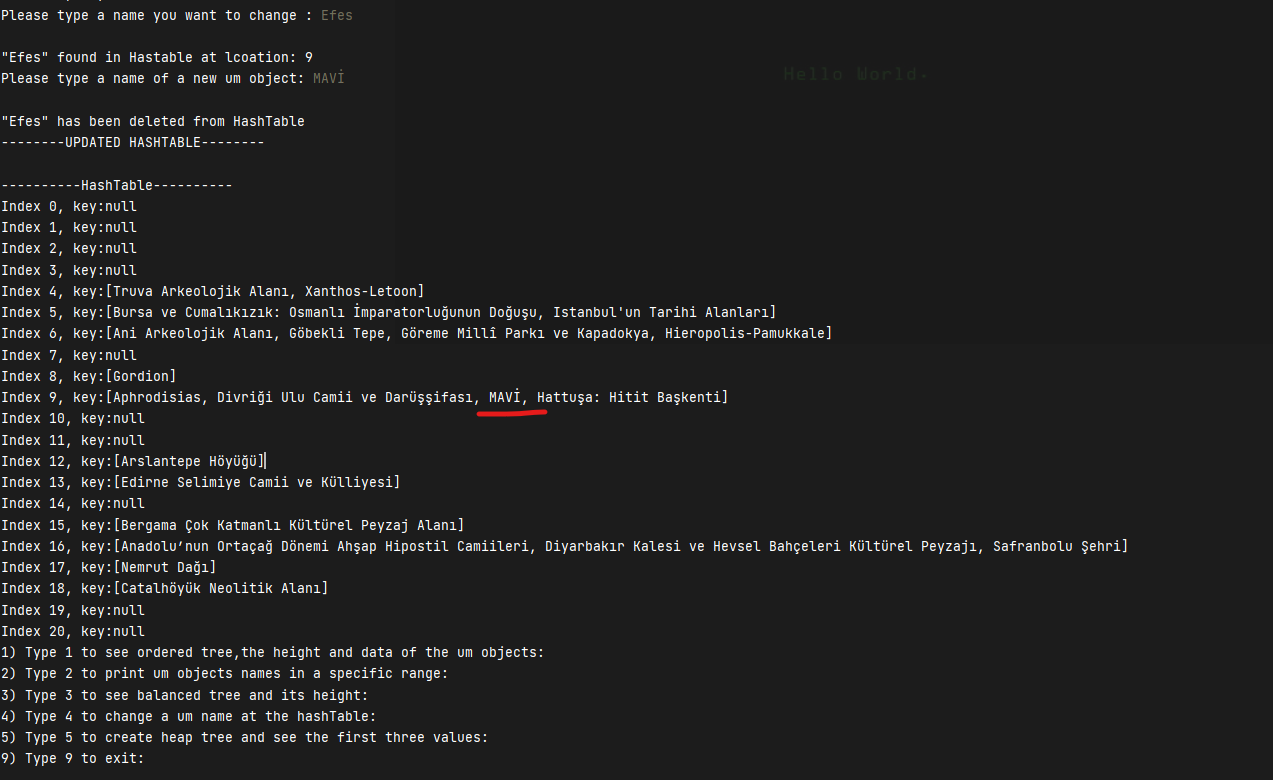
Açıklama otomatik olarak oluşturuldu

## 2.b Hash Table Güncelleme

### 2.b.1 Kaynak Kod

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***public boolean*** searchHashTable(String word) {  ***int*** newIndex = modASCIIHashFunction(word, hashTable.length);  ***if*** (hashTable[newIndex] != ***null*** && hashTable[newIndex].contains(word)) {  System.out.println("\n" + "\"" + word + "\"" + " found in Hastable at lcoation: " +newIndex);  ***return true***;  } ***else*** {  System.out.println("\n" + "\"" + word + "\"" + " not found in HashTable");  ***return false***;  }  }   ***public void*** deleteKeyHashTable(String word) {  ***int*** newIndex = modASCIIHashFunction(word, hashTable.length);  ***boolean*** result = searchHashTable(word);  ***if*** (result == ***true***) {  ***int*** index = hashTable[newIndex].indexOf(word);  hashTable[newIndex].remove(word);  System.out.print("Please type a name of a new um object: ");  Scanner scanner = ***new*** Scanner(System.in);  String inputStr = scanner.nextLine();  hashTable[newIndex].add(index,inputStr);  System.out.println("\n" + "\"" + word + "\"" + " has been deleted from HashTable ");  }  }  } |

### 2.b.2 Ekran Görüntüsü

//Güncellemenin nasıl gerçekleştiğini gösteren konsol çıktısı görüntüsü

### 2.b.3 Açıklama

//Öncesinde değiştirilmesi istenen um alanının adını kullanıcıdan istiyoruz. Eğer isim hashTable içinde mevcutsa “public boolean searchHashTable(String word)” metodu true döndürür. Bu metot doğru olduğu için ”public void deleteKeyHashTable(String word)” metodunda istenilen işlemleri yapabiliriz. İstenilen değerin arraydeki indeksini buluruz. Daha sonra o indeksteki linkedlisti dolaşırız. Mevcut alan ismi silinir ve kullanıcın yazdığı um alanının ismi hashTable içine eklenir.

# 3) HEAP VERİ YAPISI (SINIFI)

## 3.a Heap

### 3.a.1 Kaynak Kod

|  |
| --- |
| ***package*** proje33333deneme;  ***public class*** BinaryHeapPr {  UmAlani arr[];  ***int*** sizeOfTree;   ***public*** BinaryHeapPr(***int*** size) {  arr = ***new*** UmAlani[size+1];  ***this***.sizeOfTree = 0;  System.out.println("Binary Heap has been created");  }   ***public boolean*** isEmpty() {  ***if*** (sizeOfTree == 0) {  ***return true***;  } ***else*** {  ***return false***;  }  }   ***public*** String peek() {  ***if*** (isEmpty()) {  System.out.println("Binary Heap is Empty");  ***return null***;  }  ***return*** arr[1].toString();   }   ***public int*** sizeOfBP() {  ***return*** sizeOfTree;  }   ***public void*** levelOrder() {  ***for*** (***int*** i =1; i<=sizeOfTree; i++) {  System.out.println(arr[i].name + ", ");   }  }    ***public void*** heapifyBottomToUp(***int*** index){   ***int*** parent = index / 2 ;  ***if***(index <= 1){  ***return***;  }  ***else if***(arr[parent].name.compareTo(arr[index].name) > 0 ){  UmAlani temp = arr[index];  arr[index] = arr[parent];  arr[parent] = temp;   heapifyBottomToUp(parent);  }    }    ***public void*** insert(UmAlani umAlani){  arr[sizeOfTree+1] = umAlani;  sizeOfTree++;  heapifyBottomToUp(sizeOfTree);   }          } |

## 3.b Min Heap’e Yerleştirme

### 3.b.1 Kaynak Kod

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  ***public void*** heapifyBottomToUp(***int*** index){   ***int*** parent = index / 2 ;  ***if***(index <= 1){  ***return***;  }  ***else if***(arr[parent].name.compareTo(arr[index].name) > 0 ){  UmAlani temp = arr[index];  arr[index] = arr[parent];  arr[parent] = temp;   heapifyBottomToUp(parent);  }   }   ***public void*** insert(UmAlani umAlani){  arr[sizeOfTree+1] = umAlani;  sizeOfTree++;  heapifyBottomToUp(sizeOfTree);  } |

## 3.c Min Heap Listeleme

### 3.c.1 Kaynak Kod

|  |
| --- |
| //İlgili kod parçasını buraya ekleyiniz  System.out.println("BINARY HEAP"); BinaryHeapPr bh = ***new*** BinaryHeapPr(22);  ***for***(***int*** i = 0; i < 3; i++){  bh.insert(umArray[i]); } bh.levelOrder(); |

### 3.c.2 Ekran Görüntüsü

// Konsol çıktısına ait ekran görüntülerini buraya ekleyinizmetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

### 3.c.3 Açıklama

// Altyapıda kullanılan veri yapısını ve heap’ i nasıl oluşturduğunuzu açıklayınız.

BinaryHeapPr Sınıfı:

Alanlar (Fields):

arr: Minimum binary heap içindeki öğeleri depolamak için kullanılan UmAlani tipinde bir dizi.

sizeOfTree: Heap içindeki eleman sayısını tutan bir tamsayı.

Yapıcı Metot (Constructor):

arr dizisi ve sizeOfTree alanı başlatılır. size parametresi, heap boyutunu belirtir.

isEmpty Metodu:

Heap'in boş olup olmadığını kontrol eder.

peek Metodu:

Heap'in en küçük elemanını (kök) döndürür. Heap boşsa null değeri döndürür.

sizeOfBP Metodu:

Heap'in eleman sayısını döndürür.

levelOrder Metodu:

Heap'in seviye sırasında (level-order) gezilerek elemanları ekrana yazdırır.

heapifyBottomToUp Metodu:

Yeni bir eleman eklediğinizde, bu elemanın yukarıya doğru doğru sıralanmasını sağlayan bir yardımcı metottur.

Eleman eklenir ve parent'ı ile karşılaştırılır. Eğer parent'ı daha büyükse, yer değiştirilir ve bu işlem parent'lar boyunca tekrarlanır.

insert Metodu:

Heap'e yeni bir eleman ekler.

Eleman eklenir ve heapifyBottomToUp metodu ile heap yeniden düzenlenir.

Veri Yapısı ve Heap Oluşturma İşlemi:

arr dizisi, UmAlani tipindeki elemanları depolamak için kullanılır.

Minimum binary heap, her düğümün iki alt düğümü arasında belirli bir sıralamayı koruyan bir tam ağaçtır.

Yeni bir eleman eklendiğinde, eleman dizinin sonuna eklenir ve heapifyBottomToUp metodu ile yukarı doğru sıralama gerçekleştirilir.

Bu sınıf, minimum binary heap yapısını kullanarak elemanları küçükten büyüğe doğru sıralar ve işler. Eleman eklemek ve en küçük elemana erişmek gibi temel operasyonları destekler.

Bu sınıf, minimum binary heap yapısını uygulayarak öncelikli kuyruk benzeri bir veri yapısı sağlar.

# 4) SIRALAMA

## 4.a Sıralama Algoritmaları

### 4.a.1 Kaynak Kod

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace soru4  {  internal class Program  {  static void Main(string[] args)  {  int[] numbers = { 65, 24, 13, 21, 11, 3, 57, 8, 43, 33 };  Console.WriteLine("Sıralanmamış Dizi:");  PrintArray(numbers);  // Selection Sort'u uygula  SelectionSort(numbers);  Console.WriteLine("\nSıralanmış Dizi:");  PrintArray(numbers);  }  static void SelectionSort(int[] arr)  {  int n = arr.Length;  for (int i = 0; i < n - 1; i++)  {  // Minimum elemanın indisini bul  int minIndex = i;  for (int j = i + 1; j < n; j++)  {  if (arr[j] < arr[minIndex])  {  minIndex = j;  }  }  // Minimum elemanı bulunan ile değiştir  Swap(arr, i, minIndex);  // Turun sonucunu yazdır  Console.WriteLine($"Tur {i + 1}: {string.Join(", ", arr)}");  }  }  static void Swap(int[] arr, int index1, int index2)  {  int temp = arr[index1];  arr[index1] = arr[index2];  arr[index2] = temp;  }  static void PrintArray(int[] arr)  {  Console.WriteLine(string.Join(", ", arr));  }  }  } |

### 4.a.2 Açıklama

Program, sıralanacak olan tamsayı dizisini "numbers" adlı bir dizi olarak tanımlar.

PrintArray Metodu bir diziyi ekrana yazdırmak için kullanılır. Diziyi virgülle ayrılmış bir şekilde ekrana yazdırmak için string.Join metodu kullanılmıştır.

SelectionSort Metodu Selection Sort algoritmasını uygular. Dıştaki döngü, diziyi sıralamanın her bir turunu temsil eder. İçteki döngü, minimum elemanın indisini bulmaya yöneliktir. Minimum eleman bulunduktan sonra Swap metodu ile mevcut elemanla değiştirilir. Her turun sonunda dizinin o anki durumu ekrana yazdırılır.

Swap Metodu dizideki iki elemanın yerlerini değiştirmek için kullanılır. Geçici bir değişken aracılığıyla swap işlemi gerçekleştirilir.

Programın başlangıcını kontrol eden Main metodu, diziyi sıralamak için SelectionSort metodunu çağırır ve sonuçları ekrana yazdırır.

## 4.b Zaman Karmaşıklığının Hesaplanması

### 4.b.1 Açıklama

Selection Sort algoritmasının zaman karmaşıklığı O(n^2)'dir. Burada n dizinin eleman sayısını temsil eder. Her bir elemanın uygun konuma yerleştirilmesi için dizinin tamamı boyunca döngüler kullanılır. Bu nedenle, en kötü durumda (dizi tamamen ters sıralanmışsa) bile n^2 karşılaştırma ve yer değiştirme işlemi gerçekleştirilir.

Kıyaslamak için başka bir sıralama algoritması olan QuickSort'un ortalama zaman karmaşıklığı O(nlogn)’dir. QuickSort, bölünerek ve birleştirilerek çalışır. Her adımda bir eleman (pivot) seçilir, diğer elemanlar pivot'a göre sıralanır ve ardından pivot doğru konumuna yerleştirilir. Bu işlem diziyi iki parçaya böler. Bu bölme işlemi O(n) zaman alır ve algoritma bu işlemi logn kez tekrarlar.

QuickSort'un ortalama zaman karmaşıklığı O(nlogn)'dir. Bu, algoritmanın genellikle daha hızlı çalıştığı anlamına gelir, çünkü her bir bölme işlemi logaritmik sayıda adım gerektirir.

Selection Sort ise her zaman O(n^2) zaman karmaşıklığına sahiptir, bu da algoritmanın performansının veri setinin boyutuyla karesel olarak arttığı anlamına gelir.

Bu nedenle, büyük veri setleri üzerinde QuickSort genellikle daha etkilidir. Ancak, Selection Sort'un bazı durumlarda avantajlı olabilir, özellikle küçük veri setleri veya sıralı veri setleri gibi durumlarda.

Sonuç olarak, QuickSort genelde daha hızlıdır, ancak seçilecek en iyi algoritma veri setinin özelliklerine ve boyutuna bağlıdır.

# 5) Görselleştirme

## 5.a Görselleştirmenin Etkisi

### 5.a.1 Açıklama

Görselleştirme, algoritmalar ve veri yapıları konularındaki karmaşıklığı azaltarak öğrenmeyi kolaylaştırır. Soyut kavramları somutlaştırarak yeni başlayanlara bu konuları anlama olanağı sağlar. Ayrıca, farklı algoritmaların veya veri yapılarının performansını hızlıca karşılaştırmayı ve analiz etmeyi kolaylaştırır. Görselleştirmeler, algoritma veya veri yapıları üzerinde hataları tespit etmeyi ve düzeltmeyi kolaylaştırır. Kullanıcılar, görselleştirmeler aracılığıyla algoritmaları veya veri yapılarını etkileşimli bir şekilde keşfedebilirler, bu da öğrenme sürecini daha etkileşimli ve eğlenceli hale getirir. Görselleştirmeler aynı zamanda algoritma veya veri yapısının çalışma sürecini adım adım takip etmeyi kolaylaştırarak, bu konuların daha iyi anlaşılmasını sağlar. Karmaşık konuların daha sade bir şekilde görselleştirilmesi, öğrenme sürecini erişilebilir kılar. Ayrıca, görselleştirmelerin tasarımı ve analizi, bu becerileri geliştirmek isteyenlere pratik sağlar ve öğrenilen bilgilerin hafızada kalmasını artırabilir.

### 5.b Karşılaştırma Yapılması

### 5.b.1 Açıklama

Videolar ve etkileşimli görselleştirme araçları, öğrenme sürecine farklı katkılar sağlar. Videolar, genellikle izleyicilere pasif bir öğrenme deneyimi sunar ve belirli bir konuyu detaylı bir şekilde açıklama avantajına sahiptir. Ancak, videolar etkileşimli değildir ve izleyiciler, öğrenilen bilgileri uygulama veya test etme şansına sahip değillerdir. Etkileşimli görselleştirme araçları, öğrencilere konularla etkileşimde bulunma ve aktif katılım sağlama imkanı tanır. Bu araçlar, problem çözme ve uygulama yeteneklerini geliştirmek için pratik yapma şansı sunar. Etkileşimli araçlar, öğrenmeyi daha kişiselleştirilmiş ve katılımcı hale getirirken, hazırlık gereksinimleri ve standartlaştırma sorunları gibi zorluklarla karşılaşabilir. Bu bağlamda, ideal bir öğrenme deneyimi genellikle videoların sağladığı detaylı anlatımı etkileşimli araçların sağladığı etkileşim ve uygulama fırsatlarıyla birleştirmeyi gerektirebilir.

# 6) Özdeğerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 3 Maddeleri** | **Not** | **Tahmini Not** | **Açıklama** |
| **1 a) Ağaç (UNESCO M. İkili Arama Ağacı)** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **1 b) Derinlik Bulma, Ağacı Listeleme, Düğüm Sayısı Buldurma, Dengeli Ağaç Derinliği Hesaplama** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **1 c) Arama ve Listeleme** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **1 d) Kelime Ağacı Oluşturarak Kelimeleri Sayma** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **2) Hash Tablosu** | **15** | **15** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **3) Yığın Ağacı (Heap)** | **15** | **15** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **4) Sıralama Algoritmaları** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **5.a) Görselleştirmenin DSA öğrenmeye etkisi** | **5** | **5** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **5.b) Video & Görselleştirme Araçları Karşılaştırma** | **5** | **5** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |
| **6) Özdeğerlendirme Tablosu** | **10** | **10** | **Soruda nasıl yapılması istenildiyse öyle yapıldı** |

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı veya neden yapılamadığı kısaca yazılmalıdır. Tahmini not kısmına da ilgili maddeden kaç almayı beklediğinizi yazmalısınız.**

**Not: Raporu teslim edilmeyen projeler değerlendirmeye alınmayacaktır.**